```
// IEHooker.idl : IDL source for IEHooker.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (IEHooker.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
      [
            object,
            uuid(16122F01-9713-11D3-9744-005004116944),
            dual,
            helpstring("ICIEHooker Interface"),
            pointer_default(unique)
      ]
      interface ICIEHooker : IDispatch
      {
      };

[
      uuid(16122EF1-9713-11D3-9744-005004116944),
      version(1.0),
      helpstring("IEHooker 1.0 Type Library")
]
library IEHOOKERLib
{
      importlib("stdole32.tlb");
      importlib("stdole2.tlb");

      [
            uuid(16122F03-9713-11D3-9744-005004116944),
            helpstring("_ICIEHookerEvents Interface")
      ]
      dispinterface _ICIEHookerEvents
      {
            properties:
            methods:
            [id(1), helpstring("method MakeCall")] HRESULT MakeCall(BSTR
bstr);
      };

      [
            uuid(16122F02-9713-11D3-9744-005004116944),
            helpstring("CIEHooker Class")
      ]
      coclass CIEHooker
      {
            [default] interface ICIEHooker;
            [default, source] dispinterface _ICIEHookerEvents;
      };
};
```

```
#ifndef CHECK_NUM_H
#define     CHECK_NUM_H

#include "mshtml.h"

#define DO_IDLE                 0
#define DO_START        1
#define DO_CONTINUE             2
#define DO_SUCCESSEND   3
#define DO_FAILEND              4
typedef struct DATA_tag
{
      char* input_buf;
      int input_size;
      char* output_buf;
      int output_size;
      int iCheckStatus;
      BOOL bUserStop;

      struct DATA_tag*  pNext;
      IHTMLDocument2* pDocument;
}STRUCT_PROCESS;

void checkPhoneNumThread(void *pVoid);
void checkPhoneNumProc(STRUCT_PROCESS *pData);
BOOL checkPhoneNum(STRUCT_PROCESS *pData);

#endif
```

```
#include "stdafx.h"
#include "checkNum.h"

#define MIN_PHONENUM_COUNT 10

static char* szNumMapTable[] = {
      "(###)###-####",
      "(###) ###-####",
      "###.###.####",
      "###/###/####",
      "###/###-####",
      "###-###-####",
      "(###-###-####)",
      "#-###-###-####",
      "+# (###) ###-####",
      "+# (###)###-####",
      "+# ### ### ####",
      "+#-### ### ####",
      "+#-### ###-####",
      "+#-###-###-####",
      "### ### ####",
      "### ###.####",
      "### ###-####",
      "##########"
};

static const MAX_MAP_COUNT = sizeof( szNumMapTable )/sizeof( szNumMapTable[0] );

static char szInsertString1[] = {
      "<Font CLASS='PHONE' STYLE='CURSOR=HAND;' COLOR='#66CC00'
OnMouseOut=window.event.srcElement.style.color='#66CC00'
OnMouseOver=window.event.srcElement.style.color='#FF0000'
OnClick=window.open('http://www.dialpad.com/dialpad/launch.php3?Number="//cgi-
bin/launch.pl?Number="
};
static char szInsertString2[] = {
      "','Dialpad','scrollbars=no,resizable=no,width=420,height=370')><U>"
};
static char szInsertString3[] = {
      "</U></Font>"
};

static const int nInsertLen = strlen(szInsertString1) + strlen(szInsertString2)
+ strlen(szInsertString3);

BOOL IsValidNum(char* phoneNum)
{
      char num[MAX_PATH];
      strcpy(num,phoneNum);
      int nDigitCount = 0;    // track digits because since we also consider
letters
                              // in addition to numbers we need to
make sure that there
                              // is at least three digits (for area
code)

      if(strlen(phoneNum) >= MIN_PHONENUM_COUNT)
```

```
	{
		for(int i = 0 ; i < (int)strlen(num) ; i++)
		{
			if( (num[i] >= '0' && num[i] <= '9') )
			{
				nDigitCount++;
				num[i] = '#';
			}
			else if ( (num[i] >= 'A' && num[i] <= 'Z') )
			{
				num[i] = '#';
			}
		}
		for(i = (int)strlen(num) - 1 ; i >= 0  ; i--)
		{
			if(num[i] == '#') break;
			if( num[i] == ' ' ) num[i] = NULL;
		}
		for(i = 0 ; i < MAX_MAP_COUNT ; i++)
		{
			if( (strcmp(szNumMapTable[i],num) == 0)
				&& nDigitCount >= 3 )
			{
				return TRUE;
			}
		}
	}
	return FALSE;
}

BOOL isLinkTag( char* strTag)
{
	if(strcmp(strTag,"a") == 0 || strcmp(strTag,"A") == 0)
		return TRUE;
	return FALSE;

}

BOOL isLinkEndTag( char* strTag)
{
	if(strcmp(strTag,"/a") == 0 || strcmp(strTag,"/A") == 0)
		return TRUE;
	return FALSE;

}

BOOL  checkPhoneNum(STRUCT_PROCESS *pData)
{
	char *copyText;
	char ch;
	BOOL bRtn    = FALSE;
	BOOL bSkip   = FALSE;
	BOOL bNum    = FALSE;

	BOOL bLinkTag = FALSE;
	BOOL bTag     = FALSE;
```

```
	int checkSpace = nInsertLen * 10;//strlen(szInsertString1) * 10;

	char phoneNum[MAX_PATH];
	char dummyPhoneNum[MAX_PATH];
	char strTag[MAX_PATH];

	if(pData->iCheckStatus != DO_START) return FALSE;

	copyText                = new char[pData->input_size + checkSpace];
	pData->output_buf = new char[pData->input_size + checkSpace];

	int insize = pData->output_size = pData->input_size;

	memset(phoneNum,NULL,MAX_PATH);
	memset(dummyPhoneNum,NULL,MAX_PATH);
	memset(copyText,NULL,insize);
	memset(strTag,NULL,MAX_PATH);

	int j = 0;
	int k = 0;
	int l = 0;
	int m = 0;

	pData->iCheckStatus = DO_CONTINUE;

	for(int i = 0 ; i < insize ; i++) {
		if(pData->bUserStop) {
			bRtn = FALSE;
			break;
		}

		ch = pData->input_buf[i];

		if ( ( ch >= '0' && ch <= '9' )
		     || ( ch >= 'A' && ch <= 'Z' && TRUE == bNum ) )
		{
			if( bSkip == FALSE ) {
				phoneNum[j++] = ch;
				dummyPhoneNum[l++] = ch;
				bNum = TRUE;
			} else {
				copyText[k++] = ch;
			}
		}
		else
		{
			switch(ch) {
			case '.':
				if( bSkip == FALSE ) {
					if(bNum && (l == 3 || l == 7)) {
						dummyPhoneNum[l++] = ch;
					}
					// the '.' might be a sentence terminator
					else if (bNum && (l == 8 || l == 12 || l == 14))
					{
						goto check_routine;
					}
```

```
                        else {
                              for(int i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                    copyText[k++] = dummyPhoneNum[i];
                              }
                              copyText[k++] = ch;
                              bNum = FALSE;
                              memset(dummyPhoneNum,NULL,MAX_PATH);
                              memset(phoneNum,NULL,MAX_PATH);
                              j = 0;
                              l = 0;
                        }
                  } else {
                        copyText[k++] = ch;
                  }
                  break;
            case '/':
                  if( bSkip == FALSE ) {
                        if(bNum && (l == 3 || l == 7)) {
                              dummyPhoneNum[l++] = ch;
                        } else {
                              for(int i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                    copyText[k++] = dummyPhoneNum[i];
                              }
                              copyText[k++] = ch;
                              bNum = FALSE;
                              memset(dummyPhoneNum,NULL,MAX_PATH);
                              memset(phoneNum,NULL,MAX_PATH);
                              j = 0;
                              l = 0;
                        }
                  } else {
                        copyText[k++] = ch;
                  }
                  break;

            case ')':
                  if( bSkip == FALSE ) {
                        if(bNum) {
                              if(l == 4 || l == 7 || l == 13)
                                    dummyPhoneNum[l++] = ch;
                              else
                                    goto check_routine;
                        } else {
                              for(int i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                    copyText[k++] = dummyPhoneNum[i];
                              }
                              copyText[k++] = ch;
                              bNum = FALSE;
                              memset(dummyPhoneNum,NULL,MAX_PATH);
                              memset(phoneNum,NULL,MAX_PATH);
                              j = 0;
                              l = 0;
                        }
                  } else {
```

```
                        copyText[k++] = ch;
                    }
                    break;

                case '-':
                    if( bSkip == FALSE ) {
                        if(bNum)
                            dummyPhoneNum[l++] = ch;
                        else
                            copyText[k++] = ch;
                    } else {
                        copyText[k++] = ch;
                    }
                    break;
                case ' ':
                    if( bSkip == FALSE ) {
                        // check if ' ' is a terminator for the number
                        if (bNum && (l > 10))
                        {
                            goto check_routine;
                        }
                        else if(bNum)
                            dummyPhoneNum[l++] = ch;
                        else
                            copyText[k++] = ch;
                    } else {
                        copyText[k++] = ch;
                    }
                    break;
                case '(':
                case '+':
                    if( bSkip == FALSE ) {
                        if((insize >= i + 1) &&
                            pData->input_buf[i+1] >= '0' && pData-
>input_buf[i+1] <= '9') {
                            dummyPhoneNum[l++] = ch;
                            bNum = TRUE;
                        } else {
                            copyText[k++] = ch;
                        }
                    } else {
                        copyText[k++] = ch;
                    }
                    break;

                case '>':
                    if ( FALSE == bLinkTag )
                    {
                        bSkip = FALSE;
                    }
                    bTag = FALSE;
                    //if(bNum)
                    //    dummyPhoneNum[l++] = ch;
                    //else ------->why? 3/30/200
                    copyText[k++] = ch;
                    break;
```

```
                default:
      check_routine:
                        if( bSkip == FALSE ) {
                              if(bNum) {
                                    if(   bLinkTag == TRUE) {
                                          for(int i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                                copyText[k++] =
dummyPhoneNum[i];
                                          }
                                          memset(dummyPhoneNum,NULL,MAX_PATH);
                                          l = 0;
                                          copyText[k++] = ch;
                                    } else {
                                          if(!IsValidNum(dummyPhoneNum)) {
                                                for(int i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                                      copyText[k++] =
dummyPhoneNum[i];
                                                }
      memset(dummyPhoneNum,NULL,MAX_PATH);
                                                l = 0;
                                                copyText[k++] = ch;
                                          } else {
                                                for(int i = 0; i <
(int)strlen(szInsertString1) ; i++) {
                                                      copyText[k++] =
szInsertString1[i];
                                                }
                                                for(i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                                      if(dummyPhoneNum[i] >= '0'
&& dummyPhoneNum[i] <= '9')
                                                            copyText[k++] =
dummyPhoneNum[i];
                                                }
                                                for(i = 0; i <
(int)strlen(szInsertString2) ; i++) {
                                                      copyText[k++] =
szInsertString2[i];
                                                }
                                                for(i = 0 ; i <
(int)strlen(dummyPhoneNum) ; i++) {
                                                      copyText[k++] =
dummyPhoneNum[i];
                                                }
      memset(dummyPhoneNum,NULL,MAX_PATH);
                                                l = 0;
                                                for(i = 0; i <
(int)strlen(szInsertString3) ; i++) {
                                                      copyText[k++] =
szInsertString3[i];
                                                }
                                                copyText[k++] = ch;
```

```
                                        bRtn = TRUE;
                                    }
                                }
                                memset(phoneNum,NULL,MAX_PATH);
                                bNum = FALSE;
                                j = 0;
                            }
                            else
                            {
                                // bNum is false
                                bLinkTag = FALSE;
                                copyText[k++] = ch;
                            }

                            if(ch == '<') {
                                memset(strTag,NULL,MAX_PATH);
                                int idx = 1;
                                while (1) {
                                    if((insize >= i + idx) &&
                                        (pData->input_buf[i + idx] ==
'>' || pData->input_buf[i + idx] == ' '))
                                    {
                                        if(isLinkTag(strTag))
                                            bLinkTag = TRUE;
                                        else if(isLinkEndTag(strTag))
                                            bLinkTag = FALSE;
                                        break;
                                    } else {
                                        if(insize >= i + idx)
                                            strTag[idx-1] = pData-
>input_buf[i + idx];
                                        else
                                            break;
                                    }
                                    idx++;
                                }
                                bSkip = TRUE;
                            }
                        } else { // bSkip is true
                            copyText[k++] = ch;

                            // reset bLinkTag if reached end anchor
                            if(ch == '<') {
                                memset(strTag,NULL,MAX_PATH);
                                int idx = 1;
                                while (1) {
                                    if((insize >= i + idx) &&
                                        (pData->input_buf[i + idx] ==
'>' || pData->input_buf[i + idx] == ' '))
                                    {
                                        if(isLinkTag(strTag))
                                            bLinkTag = TRUE;
                                        else if(isLinkEndTag(strTag))
                                            bLinkTag = FALSE;
                                        break;
                                    } else {
                                        if(insize >= i + idx)
```

```
                                                strTag[idx-1] = pData-
>input_buf[i + idx];
                                          else
                                                break;
                                    }
                                    idx++;
                              }
                        }
                  }
                  break;
            }
      }

      if( (k + checkSpace) >= pData->output_size) {
            int resize;
            resize = pData->output_size + checkSpace;
            strcpy(pData->output_buf,copyText);
            delete[] copyText;
            copyText=new char[resize];
            memset(copyText,NULL,resize);
            strcpy(copyText,pData->output_buf);
            delete[] pData->output_buf;
            pData->output_buf = new char[resize];
            memset(pData->output_buf,NULL,resize);
            pData->output_size = resize;
      }
}

if(bNum) {
      if( k + checkSpace >= pData->output_size) {
            int resize;
            resize = pData->output_size + checkSpace;
            strcpy(pData->output_buf,copyText);
            delete[] copyText;
            copyText=new char[resize];
            memset(copyText,NULL,resize);
            strcpy(copyText,pData->output_buf);
            delete[] pData->output_buf;
            pData->output_buf = new char[resize];
            memset(pData->output_buf,NULL,resize);
            pData->output_size = resize;
      }
      if(!IsValidNum(phoneNum)) {
            for(int n = 0 ; n < (int)strlen(dummyPhoneNum) ; n++) {
                  copyText[k++] = dummyPhoneNum[n];
            }
            memset(dummyPhoneNum,NULL,MAX_PATH);
      } else {    // phone number is invalid
            for(int i = 0; i < (int)strlen(szInsertString1) ; i++) {
                  copyText[k++] = szInsertString1[i];
            }
            for(i = 0 ; i < (int)strlen(dummyPhoneNum) ; i++) {
                  if(dummyPhoneNum[i] >= '0' && dummyPhoneNum[i] <= '9')
                        copyText[k++] = dummyPhoneNum[i];
            }
            for(i = 0; i < (int)strlen(szInsertString2) ; i++) {
                  copyText[k++] = szInsertString2[i];
```

```
                }
                for(i = 0 ; i < (int)strlen(dummyPhoneNum) ; i++) {
                        copyText[k++] = dummyPhoneNum[i];
                }
                for(i = 0; i < (int)strlen(szInsertString3) ; i++) {
                        copyText[k++] = szInsertString3[i];
                }
                bRtn = TRUE;
            }
        }

        if(bRtn == TRUE) {
            delete[] pData->output_buf;
            pData->output_buf = new char[k + 1];//-1];
            memset(pData->output_buf,NULL,k + 1);//-1);
            memcpy(pData->output_buf,copyText,k-1);//2);
        }
        delete[] copyText;
        return bRtn;
}

void checkPhoneNumThread(void *pVoid)
{
        STRUCT_PROCESS *pData =(STRUCT_PROCESS *)pVoid;

        if(checkPhoneNum(pData))
            pData->iCheckStatus = DO_SUCCESSEND;
        else
            pData->iCheckStatus = DO_FAILEND;
}

void checkPhoneNumProc(STRUCT_PROCESS *pData)
{
        HANDLE hThread;
        DWORD nID;

        pData->iCheckStatus = DO_START;

        hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,        // No security attributes.
                    (DWORD)0,                           // Use same stack size.
                    (LPTHREAD_START_ROUTINE)checkPhoneNumThread, // Thread
procedure.
                    (LPVOID)pData,                                  // Parameter
to pass.
                    (DWORD)0,                           // Run immediately.
                    (LPDWORD)&nID);
        CloseHandle(hThread);
}
```

```
// CIEHooker.h : Declaration of the CCIEHooker

#ifndef __CIEHOOKER_H_
#define __CIEHOOKER_H_

#include "resource.h"       // main symbols
#include "IEHookerCP.h"
#include "ExDispID.h"
#include "mshtmdid.h"
#include "mshtml.h"
#include <strstrea.h>

#include "checkNum.h"

/////////////////////////////////////////////////////////////////////////////
// CCIEHooker
class ATL_NO_VTABLE CCIEHooker :
      public CComObjectRootEx<CComSingleThreadModel>,
      public CComCoClass<CCIEHooker, &CLSID_CIEHooker>,
      public IObjectWithSiteImpl<CCIEHooker>,
      public ISupportErrorInfo,
      public IConnectionPointContainerImpl<CCIEHooker>,
      public IDispatchImpl<ICIEHooker, &IID_ICIEHooker, &LIBID_IEHOOKERLib>,
      public CProxy_ICIEHookerEvents< CCIEHooker >
{
public:

DECLARE_REGISTRY_RESOURCEID(IDR_CIEHOOKER)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CCIEHooker)
      COM_INTERFACE_ENTRY(ICIEHooker)
      COM_INTERFACE_ENTRY(IDispatch)
      COM_INTERFACE_ENTRY(ISupportErrorInfo)
      COM_INTERFACE_ENTRY(IConnectionPointContainer)
      COM_INTERFACE_ENTRY(IObjectWithSite)
      COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
END_COM_MAP()
BEGIN_CONNECTION_POINT_MAP(CCIEHooker)
      CONNECTION_POINT_ENTRY(DIID__ICIEHookerEvents)
END_CONNECTION_POINT_MAP()

public:
      CCIEHooker();
      ~CCIEHooker();

      //
      // ISupportsErrorInfo
      //
      STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

   //
   // IDispatch Methods
   //
   STDMETHOD(Invoke)(DISPID dispidMember,REFIID riid, LCID lcid, WORD wFlags,
                     DISPPARAMS * pdispparams, VARIANT * pvarResult,
```

```
                    EXCEPINFO * pexcepinfo, UINT * puArgErr);

   //
   // IOleObjectWithSite Methods
   //
      STDMETHOD(SetSite)(IUnknown *pUnkSite);

private:
      DWORD m_dwCookie;   // Connection Token - used for Advise and Unadvise
      CComQIPtr<IWebBrowser2, &IID_IWebBrowser2> m_spWebBrowser2;
      enum ConnectType { Advise, Unadvise };   // What to do when managing the
connection

      BOOL ManageConnection(enum ConnectType eConnectType);

      void checkAllData();
    void checkUpdate(STRUCT_PROCESS* pData);
      void EnumFrames();
      void RecurseWindows(IHTMLDocument2* pDocument);

      BOOL GetPageBody(IHTMLDocument2* pDocument);
      BOOL AddData(STRUCT_PROCESS* pData);
      BOOL RemoveAll();
      BOOL RemoveData(STRUCT_PROCESS* pData);

public:
};

#endif //__CIEHOOKER_H_
```

```
// CIEHooker.cpp : Implementation of CCIEHooker
#include "stdafx.h"
#include "IEHooker.h"
#include "CIEHooker.h"
#include <comdef.h>

////////////////////////////////////////////////////////////////////////////
// CCIEHooker
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

STRUCT_PROCESS *g_pHeaderData = NULL;
DWORD iCurrentTick = 0;

void EnablePhoneParse(BOOL bEnablePhoneParse)
{
	HKEY  hAppKey;
	DWORD dwDisposition;

	if(RegCreateKeyEx(HKEY_LOCAL_MACHINE, TEXT("Software\\DialPad.com"), 0,
NULL,
		REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, &hAppKey, &dwDisposition)
		== ERROR_SUCCESS)
	{
		RegSetValueEx(hAppKey, TEXT("EnablePhoneParse"), 0,
REG_BINARY,(LPBYTE)&bEnablePhoneParse, sizeof(WORD));
		RegCloseKey(hAppKey);
	}
}

BOOL IsEnablePhoneParse()
{
	HKEY  hAppKey;
	DWORD bEnablePhoneParse = 0;
	DWORD ulSize;

	if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("Software\\DialPad.com"), 0,
KEY_READ,
		&hAppKey) == ERROR_SUCCESS)
	{
		ulSize = sizeof(DWORD);
		RegQueryValueEx(hAppKey, TEXT("EnablePhoneParse"), NULL,
NULL,(LPBYTE)&bEnablePhoneParse, &ulSize);
		RegCloseKey(hAppKey);
	}
	return (BOOL)bEnablePhoneParse;
}

CCIEHooker::CCIEHooker()
{
}

CCIEHooker::~CCIEHooker()
{
```

```
}

STDMETHODIMP CCIEHooker::InterfaceSupportsErrorInfo(REFIID riid)
{
/*    static const IID* arr[] =
      {
            &IID_ICIEHooker
      };
      for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
      {
            if (InlineIsEqualGUID(*arr[i],riid))
                  return S_OK;
      }
*/    return S_FALSE;
}

BOOL CCIEHooker::ManageConnection(enum ConnectType eConnectType)
{
   HRESULT hr;

   if (!m_spWebBrowser2) return S_OK;

   CComQIPtr<IConnectionPointContainer,
             &IID_IConnectionPointContainer> spCPContainer(m_spWebBrowser2);

   if (spCPContainer != NULL) {
      CComPtr<IConnectionPoint> spConnectionPoint;
      hr = spCPContainer->FindConnectionPoint(DIID_DWebBrowserEvents2,
&spConnectionPoint);
      if (SUCCEEDED(hr)) {
         if (eConnectType == Advise) {
            hr = spConnectionPoint->Advise((IDispatch*)this, &m_dwCookie);
         } else {
            hr = spConnectionPoint->Unadvise(m_dwCookie);
         }
      }
   }
   return (SUCCEEDED(hr));
}

STDMETHODIMP CCIEHooker::SetSite(IUnknown *pUnkSite)
{
      USES_CONVERSION;

      if (!pUnkSite) {
            ATLTRACE("\nSetSite(): pUnkSite is NULL\n\n");
      } else {
            m_spWebBrowser2 = pUnkSite;
            if (m_spWebBrowser2) {
                  if (!ManageConnection(Advise)) {
                        ATLTRACE("Failure sinking events from IWebBrowser2");
                  }
            }
      }
      return S_OK;
}
```

```
STDMETHODIMP CCIEHooker::Invoke(DISPID dispidMember,
                                REFIID riid,
                                LCID lcid,
                                WORD wFlags,
                                DISPPARAMS* pDispParams,
                                VARIANT* pvarResult,
                                EXCEPINFO*  pExcepInfo,
                                UINT* puArgErr)
{
   USES_CONVERSION;

   if (!pDispParams)
      return E_INVALIDARG;

   if(iCurrentTick) {
         DWORD iTmpTick = ::GetTickCount();
         if(iTmpTick - iCurrentTick > 2000) {
               iCurrentTick = 0;
               EnumFrames();
         }
   } else {
         checkAllData();
   }

   switch (dispidMember)
   {
      case DISPID_DOCUMENTCOMPLETE:
            if(IsEnablePhoneParse()) {
                  if (pDispParams->cArgs >= 2 && pDispParams->rgvarg[1].vt ==
VT_DISPATCH) {
                        if(m_spWebBrowser2 == pDispParams->rgvarg[1].pdispVal) {
                              iCurrentTick = 0;
                              RemoveAll();
                              EnumFrames();
                        } else {
                              iCurrentTick = ::GetTickCount();
                        }
                  }
            }
            break;
      case DISPID_BEFORENAVIGATE2:
            break;

      case DISPID_NAVIGATECOMPLETE2:
      case DISPID_STATUSTEXTCHANGE:
      case DISPID_PROGRESSCHANGE:
      case DISPID_DOWNLOADBEGIN:
      case DISPID_DOWNLOADCOMPLETE:
      case DISPID_COMMANDSTATECHANGE:
      case DISPID_NEWWINDOW2:
      case DISPID_BEFORENAVIGATE:
            break;

      case DISPID_ONQUIT:
            RemoveAll();
            ManageConnection(Unadvise);
            break;
```

```
      default:
            break;
   }
   return S_OK;
}

void CCIEHooker::checkAllData()
{
      STRUCT_PROCESS* pCurData = NULL;
      STRUCT_PROCESS* pNextData = NULL;

      if(g_pHeaderData) {
            try {
                  pCurData = pNextData = g_pHeaderData;
                  while(pNextData) {
                        pCurData = pNextData;
                        if(pCurData->iCheckStatus == DO_SUCCESSEND)
                              checkUpdate(pCurData);
                        pNextData = pCurData->pNext;
                  }
            } catch(...) {
                  ATLTRACE(_T("Unspecified exception thrown in
checkAllData\n"));
            }
      }
}

void CCIEHooker::checkUpdate(STRUCT_PROCESS* pData)
{
      if(pData) {
            if(pData->iCheckStatus == DO_SUCCESSEND) {
                  try {
                        pData->iCheckStatus = DO_IDLE;
                        int size = MultiByteToWideChar(CP_ACP,0,pData-
>output_buf,-1,0,0);
                        if(pData->output_size - pData->input_size >= 0) {
                              IHTMLElement* pbody = NULL;
                              IHTMLDocument2* pHtmlDocument = pData->pDocument;
                              if (pHtmlDocument) {
                                    if(SUCCEEDED(pHtmlDocument->get_body( &pbody
))) {
                                          if(pbody) {
                                                BSTR bstr;
                                                OLECHAR* olestr = new
OLECHAR[size+1];
                                                olestr[0] = 0;
                                                int irtn =
MultiByteToWideChar(CP_ACP,0,pData->output_buf,size,olestr,size);
                                                bstr = SysAllocString(olestr);
                                                pbody->put_innerHTML(bstr);
                                                SysFreeString(bstr);
                                                delete[] olestr;
                                          }
                                    }
                              }
                        }
```

```
                } catch(_com_error Error) {
                        ATLTRACE(Error.ErrorMessage());
                } catch(...) {
                        ATLTRACE(_T("Unspecified exception thrown in
checkUpdate\n"));
                }
        }
    }
}

BOOL checkChange(IHTMLDocument2* pDocument)
{
    STRUCT_PROCESS *pNextData = NULL;
    STRUCT_PROCESS *pCurData = NULL;

    if(g_pHeaderData == NULL)
        return TRUE;
    if(pDocument != g_pHeaderData->pDocument)
        return TRUE;
    pCurData = pNextData = g_pHeaderData->pNext;

    try {
        IHTMLFramesCollection2* pFrameset = NULL;
        pDocument->get_frames(&pFrameset);
        if(pFrameset) {
            IHTMLWindow2* pWindow2Next;
            IHTMLDocument2* pNextDoc;
            IHTMLElement* pbody;
            long len;
            pFrameset->get_length(&len);
            for(long i = 0; i <len ; i++) {
                _variant_t va(i, VT_I4);
                VARIANT _result;
                VariantInit(&_result);

                pFrameset->item(&va, &_result);

                pWindow2Next = (IHTMLWindow2*)_result.pdispVal;
                pWindow2Next->get_document(&pNextDoc);

                pCurData = pNextData;
                if(pCurData) {
                    if(pCurData->pDocument != pNextDoc) return TRUE;

                    pNextData = pCurData->pNext;
                    if(pNextDoc) {
                        if(checkChange(pNextDoc))     return TRUE;
                    }
                } else {
                    if(pNextDoc) {
                        if(SUCCEEDED(pNextDoc->get_body(&pbody))) {
                            if(pbody) {
                                BSTR bstr;
                                if(SUCCEEDED(pbody-
>get_innerHTML(&bstr))) {
                                    if (bstr) return TRUE;
                                }
```

```
                                        }
                                    }
                                    if(checkChange(pNextDoc))     return TRUE;
                                }
                            }
                        }
                    }
                } catch(...) {
                    ATLTRACE(_T("Unspecified exception checkChange\n"));
                }
                return FALSE;
}

void CCIEHooker::EnumFrames()
{
        IDispatch * pDisp;
        IHTMLDocument2* pDocument;
        try {
                if(SUCCEEDED(m_spWebBrowser2->get_Document(&pDisp))) {
                        if(SUCCEEDED(pDisp-
>QueryInterface(IID_IHTMLDocument2,(void**)&pDocument))) {
                                if(pDocument) {
                                        BOOL bcheck = checkChange(pDocument);
                                        if(bcheck) {
                                                RemoveAll();
                                                RecurseWindows(pDocument);
                                        }
                                } else {
                                        RemoveAll();
                                }
                        }
                }
        } catch(...) {
                ATLTRACE(_T("Unspecified exception EnumFrames\n"));
        }
}

void CCIEHooker::RecurseWindows(IHTMLDocument2* pDocument)
{
        IHTMLElement* pbody;

        if(pDocument == NULL) return;

        try {
                pDocument->get_body( &pbody );
                if(pbody) {
                        GetPageBody(pDocument);

                        // Get the IDispatch of the document
                        LPDISPATCH lpDisp = pDocument;
                        //lpDisp = m_webBrowser.GetDocument();

                        if (lpDisp)
                        {
                           IOleContainer* pContainer;

                           // Get the container
```

```
                    HRESULT hr = lpDisp->QueryInterface(IID_IOleContainer,
(void**)&pContainer);
                    lpDisp->Release();

                    if (FAILED(hr))
                    {
                         return;
                    }

                    IEnumUnknown* pEnumerator;

                    hr = pContainer->EnumObjects(OLECONTF_EMBEDDINGS,
&pEnumerator);
                    pContainer->Release();

                    if (FAILED(hr))
                    {
                         return;
                    }

                    IUnknown* pUnk;
                    ULONG uFetched;

                    // Enumerate all the frames and process their html info
                    for (UINT i = 0; S_OK == pEnumerator->Next(1, &pUnk,
&uFetched); i++)
                    {
                         IWebBrowser2* pBrowser;

                         hr = pUnk->QueryInterface(IID_IWebBrowser2,
(void**)&pBrowser);
                         pUnk->Release();

                         if (SUCCEEDED(hr))
                         {
                                   IDispatch * pDisp;
                                   IHTMLDocument2* pDocument;
                                   try {
                                        if(SUCCEEDED(pBrowser-
>get_Document(&pDisp))) {
                                             if(SUCCEEDED(pDisp-
>QueryInterface(IID_IHTMLDocument2,(void**)&pDocument))) {
                                                  if(pDocument) {

      RecurseWindows(pDocument);
                                                       pDocument-
>Release();
                                                  }
                                             }
                                        }
                                   } catch(...) {
                                        ATLTRACE(_T("Unspecified exception
EnumFrames\n"));
                                   }
                              pBrowser->Release();
                         }
```

```
                    }
                    pEnumerator->Release();
                }
            }
        } catch(...) {
            ATLTRACE(_T("Unspecified exception RecurseWindows\n"));
        }
}

BOOL CCIEHooker::GetPageBody(IHTMLDocument2* pDocument)
{
        BSTR bstr;
        IHTMLElement* pbody = NULL;
        BOOL bSuccess = FALSE;
        try {
            STRUCT_PROCESS* pData = new STRUCT_PROCESS;
            memset(pData,NULL,sizeof(STRUCT_PROCESS));

            if (SUCCEEDED(pDocument->get_body(&pbody))) {
                if(pbody) {
                    pbody->get_innerHTML(&bstr);
                    if (bstr) {
                        int bytes=WideCharToMultiByte(CP_ACP,0,bstr,-
1,0,0,0,0);
                        if (bytes) {
                            char *buf=new char[bytes];
                            memset(buf,NULL,bytes);
                            WideCharToMultiByte(CP_ACP,0,bstr,-
1,buf,bytes,0,0);

                            pData->input_buf = buf;
                            pData->input_size = bytes;
                            pData->pDocument = pDocument;
                            checkPhoneNumProc(pData);

                            bSuccess = TRUE;
                        }
                    }
                }
            }
            if(bSuccess) AddData(pData);
            else         RemoveData(pData);
        } catch(...) {
            ATLTRACE(_T("Unspecified exception thrown in GetPageBody\n"));
        }
        return TRUE;
}

BOOL CCIEHooker::AddData(STRUCT_PROCESS* pData)
{
        STRUCT_PROCESS* pCurData;
        STRUCT_PROCESS* pNextData;

        try {
            if(g_pHeaderData) {
                pCurData = pNextData = g_pHeaderData;
                while(pNextData) {
```

```
                        pCurData = pNextData;
                        pNextData = pNextData->pNext;
                    }
                    pCurData->pNext = pData;
                } else {
                    g_pHeaderData = pData;
                }
            } catch(...) {
                ATLTRACE(_T("Unspecified exception thrown in AddData\n"));
            }
            return TRUE;
}

BOOL CCIEHooker::RemoveAll()
{
            try {
                while(g_pHeaderData) {
                    STRUCT_PROCESS* pCurData = g_pHeaderData;
                    g_pHeaderData = pCurData->pNext;
                    RemoveData(pCurData);
                }
                g_pHeaderData = NULL;
            } catch(...) {
                ATLTRACE(_T("Unspecified exception thrown in RemoveAll\n"));
            }
            return TRUE;
}

BOOL CCIEHooker::RemoveData(STRUCT_PROCESS* pData)
{
            try {
                if(pData != NULL) {
                    pData->bUserStop = TRUE;
                    while(pData->iCheckStatus == DO_CONTINUE) {
                        Sleep(10);
                    }

                    if(pData->pDocument) {
                        pData->pDocument->Release();
                        pData->pDocument = NULL;
                    }
                    if(pData->input_buf) {
                        delete[] pData->input_buf;
                        pData->input_buf = NULL;
                    }
                    if(pData->output_buf) {
                        delete[] pData->output_buf;
                        pData->output_buf = NULL;
                    }
                    delete pData;
                    pData = NULL;
                }
            } catch(...) {
                ATLTRACE(_T("Unspecified exception thrown in RemoveData\n"));
            }
            return TRUE;
}
```

```
HKCR
{
      IEHooker.CIEHooker.1 = s 'CIEHooker Class'
      {
            CLSID = s '{16122F02-9713-11D3-9744-005004116944}'
      }
      IEHooker.CIEHooker = s 'CIEHooker Class'
      {
            CLSID = s '{16122F02-9713-11D3-9744-005004116944}'
            CurVer = s 'IEHooker.CIEHooker.1'
      }
      NoRemove CLSID
      {
            ForceRemove {16122F02-9713-11D3-9744-005004116944} = s 'CIEHooker
Class'
            {
                  ProgID = s 'IEHooker.CIEHooker.1'
                  VersionIndependentProgID = s 'IEHooker.CIEHooker'
                  ForceRemove 'Programmable'
                  InprocServer32 = s '%MODULE%'
                  {
                        val ThreadingModel = s 'Apartment'
                  }
                  'TypeLib' = s '{16122EF1-9713-11D3-9744-005004116944}'
            }
      }
}

HKLM
{
      SOFTWARE
      {
            Microsoft
            {
                  Windows
                  {
                        CurrentVersion
                        {
                              Explorer
                              {
                                    'Browser Helper Objects'
                                    {
                                          {16122F02-9713-11D3-9744-005004116944}
                                    }
                              }
                        }
                  }
            }
      }
}
```